# A Graph Rewriting Semantics for the Polyadic π-Calculus*

BARBARA KÖNIG
*Fakultät für Informatik, Technische Universität München*

**Abstract**

We give a hypergraph rewriting semantics for the polyadic π-calculus, based on rewriting rules equivalent to those in the double-pushout approach. The structural congruence of the π-calculus is replaced by hypergraph isomorphism. The correctness of the encoding from the graph-based notation into π-calculus can be shown by using an intermediate notation, so-called name-based graph terms.

## 1 Introduction

We present an alternative semantics for the asynchronous polyadic π-calculus, describing reductions by (hyper-)graph rewriting steps. The π-calculus [7] is a well-known calculus describing communicating processes with mobility.

The aim of this paper is to show that graph rewriting can be used as a natural semantics for the π-calculus and related calculi, and it wants to describe a precise way of connecting the string-based (or name-based) representation of processes to their graph-based representation.

While the first version of the π-calculus had a labelled transition semantics, in the last few years it became more and more common to use reduction semantics, splitting the labelled transitions into two relations: a structural congruence relating processes of the same structure and an unlabelled reduction relation. This was first proposed by Berry and Boudol with their chemical abstract machine [1] and is now perceived as a natural way of presenting the semantics of process calculi.

The idea is to go one step further, to represent processes as graphs instead of strings, and to replace structural congruence by graph isomorphism. This gives a denotational semantics to at least part of the reduction relation and can be very useful in the design of new process calculi. Especially in fusion calculus and its variants—a process calculus where a process may choose to fuse two channel names [11]—it is not obvious how the structural congruence should look like. Furthermore string-based process calculi have to deal with alpha-conversion, i.e.

*Research supported by SFB 342 (subproject A3) of the DFG.

the renaming of bound names, which often impedes the analysis of processes. Also interaction is done in a non-local way, i.e. parts of a process which are at the opposite ends of its description may interact with each other (this is partly alleviated by the reduction semantics), while in a graph rewriting semantics, interacting components are always adjacent.

Furthermore a graph-based calculus may serve as a basis for the analysis of mobile processes [4] and as a starting point for a visual concurrent programming language. We also want to demonstrate the usefulness of our graph construction operator (introduced in [5]) which can be used to reason about graphs in an inductive way.

When modelling $\pi$-calculus semantics by graph rewriting, we use hierarchical graphs (our design desicion are explained in more detail in section 4). The encoding of the graph-based semantics into the standard semantics requires an intermediate step: the representation of hypergraphs in a name-based notation (section 3) which highlights the different perception regarding connections in graph notation (explicit connections) and in string notation (implicit connections by common names).

# 2 Hypergraphs and Hypergraph Construction

We first define some basic notions concerning hypergraphs.

**Definition 1 (Hypergraph, Hypergraph Morphism, Isomorphism)** *Let $L$ be a fixed set of labels. A* hypergraph $H = (V_H, E_H, s_H, l_H, \chi_H)$ *consists of a set of nodes $V_H$, a set of edges $E_H$, a source mapping $s_H : E_H \to V_H^*$, a labelling $l_H : E_H \to L$ and a duplicate-free string $\chi_H \in V_H^*$ of external nodes. A hypergraph morphism $\phi : H \to H'$ maps nodes to nodes and edges to edges, preserving source nodes and labelling. A strong morphism (denoted by the arrow $\twoheadrightarrow$) additionally preserves the external nodes. We write $H \cong H'$ ($H$ is isomorphic to $H'$) if there is a bijective strong morphism from $H$ to $H'$.*

The arity of a hypergraph $H$ is defined as $ar(H) = |\chi_H|$ while the arity of an edge $e$ of $H$ is $ar(e) = |s_H(e)|$. We can regard hypergraphs and (strong) hypergraph morphisms as objects respectively morphisms of a category.

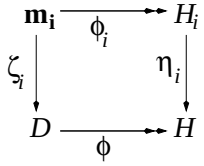
*Notation:* We call a hypergraph *discrete*, if its edge set is empty. By **m** we denote a discrete graph of arity $m \in \mathbb{N}$ with $m$ nodes where every node is external (external nodes are labelled (1), (2), ... in their respective order).

$H = edge_n(l)$ is the hypergraph with exactly one edge $e$ with label $l$ where $s_H(e) = \chi_H$, $|\chi_H| = n$ and $V_H = Set(\chi_H)$, where $Set(\tilde{s})$ is the set of all elements of a string $\tilde{s}$.

We now present a "concatenation operation" on hypergraphs. The construction plan telling us how this concatenation is supposed to happen is represented by

hypergraph morphisms mapping discrete graphs to discrete graphs. The following definitions use concepts from category theory, namely categories and colimits.

**Definition 2 (Hypergraph Construction)** *Let $H_1,\ldots,H_n$ be hypergraphs and let*[1] *$\zeta_i : \mathbf{m_i} \to D$, $i \in [n]$ be morphisms where $ar(H_i) = m_i$ and $D$ is discrete. There is always a unique strong morphism $\phi_i : \mathbf{m_i} \twoheadrightarrow H_i$ for every $i \in [n]$. Let $H$ be the colimit of $\zeta_1,\ldots,\zeta_n,\phi_1,\ldots,\phi_n$ such that $\phi$ is a strong morphism. We define: $\bigotimes_{i=1}^{n}(H_i,\zeta_i) = H$. (Alternatively we write $(H_1,\zeta_1)\otimes\ldots\otimes(H_n,\zeta_n)$—or $\otimes(H_1,\zeta_1)$, if $n=1$—instead of $\bigotimes_{i=1}^{n}(H_i,\zeta_i)$.)*




Rewriting based on our graph construction operator has the same expressive power as the double-pushout approach (with injective production spans) of Ehrig [2] (see also [5]).

# 3 A Name-based Notation for Hypergraphs

We now introduce a name-based notation for hypergraphs which will help us make the transition from the graph rewriting semantics to the standard π-calculus.

**Definition 3 (Name-based Graph Terms)** *Let $N$ be a set of names. A* name-based graph term *$h$ has one of the following forms:*

**(Empty Graph)** $0$ **(Node)** $\langle a\rangle$, $a \in N$ **(Parallel Composition)** $h_1|h_2$

**(Edge labelled $l$)** $(l)[\tilde{t}]$, $l \in L$, $\tilde{t} \in N^*$ **(Node Hiding)** $(\nu c)h$, $c \in N$

*where $h_1,h_2,h$ are again name-based graph-terms.*

*The set of free names $fn(h)$ of a term $h$ contains all names in $h$ which are not bound by $\nu$. Two graph terms are considered the same if they can be transformed into one another by consistent renaming of bound names (= α-conversion).*

*Notation:* If $\tilde{s} = a_1\ldots a_n$ is a string, then $\lfloor\tilde{s}\rfloor_i = a_i$ denotes its $i$-th element. If $M$ is a set of elements, then $\tilde{s}\cap M$ denotes the string which results from removing all elements from $\tilde{s}$ which are not in $M$.

Let $\tilde{t}$ be a duplicate-free string of names, $\tilde{s}$ an arbitrary string of the same length and $h$ a name-based graph term. Then $h\{\tilde{s}/\tilde{t}\}$ denotes the simultaneous substitution of all names $\lfloor\tilde{t}\rfloor_i$ by $\lfloor\tilde{s}\rfloor_i$. Let $\tilde{s},\tilde{t}$ be strings of names where $Set(\tilde{s}) \subseteq Set(\tilde{t})$, $n = |\tilde{s}|$, $m = |\tilde{t}|$ and $\tilde{t}$ contains no duplicates. $\zeta_{\tilde{s}-\tilde{t}} : \mathbf{n} \to \mathbf{m}$ is a morphism satisfying $\zeta_{\tilde{s}-\tilde{t}}(\lfloor\chi_{\mathbf{n}}\rfloor_i) = \lfloor\chi_{\mathbf{m}}\rfloor_j \iff \lfloor\tilde{s}\rfloor_i = \lfloor\tilde{t}\rfloor_j$.

**Definition 4 (Interpretation of Name-based Graph Terms)**

*Let $h$ be a name-based graph term and let $\tilde{t} \in N^*$ be a duplicate-free string satisfying $fn(h) = Set(\tilde{t})$. We define $\Delta_{\tilde{t}}(h)$ with:*

[1] $[n]$ stands for the set $\{1,\ldots,n\}$.

$$\Delta_\varepsilon(0) = \mathbf{0} \quad \Delta_a(\langle a\rangle) = \mathbf{1} \quad \Delta_{\tilde{t}}(h_1|\ldots|h_n) = \bigotimes_{i=1}^{n}(\Delta_{\tilde{t}_i}(h_i), \zeta_{\tilde{t}_i \to \tilde{t}}),\ \tilde{t}_i = \tilde{t} \cap fn(h_i)$$

$$\Delta_{\tilde{t}}((l)[\tilde{s}]) = \otimes(edge_{|\tilde{s}|}(l), \zeta_{\tilde{s}\to\tilde{t}}) \qquad \Delta_{\tilde{t}}((\nu c)h) = \begin{cases} \otimes(\Delta_{\tilde{t}c}(h), \zeta) & \text{if } c \in fn(h) \\ \Delta_{\tilde{t}}(h) & \text{otherwise} \end{cases}$$

*where $\zeta$ is a discrete morphism hiding the last external node of $\Delta_{\tilde{t}c}(h)$.*

There is a set of equations completely characterizing hypergraph isomorphism on name-based graph terms. The rules are strongly reminiscent of structural congruence in process calculi.

**Proposition 1 (Equivalence for Name-based Graph Terms)** *Let $\simeq$ be the smallest congruence generated by the rules below. Two terms $h, h'$ are equivalent wrt. these equations if and only if $\Delta_{\tilde{t}}(h) \cong \Delta_{\tilde{t}}(h')$.*

$$h_1|h_2 \simeq h_2|h_1 \qquad h_1|(h_2|h_3) \simeq (h_1|h_2)|h_3 \qquad (\nu c)0 \simeq 0 \qquad h|0 \simeq h$$
$$(\nu c)(\nu b)h \simeq (\nu b)(\nu c)h \qquad (l)[a_1\ldots a_n]|\langle a_i\rangle \simeq (l)[a_1\ldots a_n]$$
$$\langle c\rangle|\langle c\rangle \simeq \langle c\rangle \qquad ((\nu c)h_1)|h_2 \simeq (\nu c)(h_1|h_2) \text{ if } c \notin fn(h_2)$$

We now define and relate rewrite steps in the two notations:

**Proposition 2 (Rewriting Hypergraphs)** *Let $L, R$ be two hypergraphs of the same arity. By $\to_{(L,R)}$ we denote the smallest relation generated by $L \to_{(L,R)} R$ and closed under hypergraph construction and isomorphism.*

*Now let $l, r$ be two name-based graph terms satisfying $fn(l) = fn(r) = Set(\tilde{t})$ where $\tilde{t} \in N^*$ contains no duplicates. The relation $\to_{(l,r)}$ is generated by the rule $l\{\tilde{s}/\tilde{t}\} \to_{(l,r)} r\{\tilde{s}/\tilde{t}\}$ and is closed under context and $\simeq$.*

*It holds that $h \to_{(l,r)} h'$ if and only if $\Delta_{\tilde{s}}(h) \to_{(\Delta_{\tilde{t}}(l),\Delta_{\tilde{t}}(r))} \Delta_{\tilde{s}}(h')$ (where $\tilde{s} \in N^*$ contains exactly the free names of $h$).*

# 4 Process Graphs

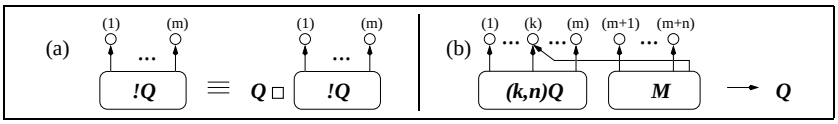
We show how to define a graph rewriting semantics for the asynchronous polyadic $\pi$-calculus [7] with so-called process graphs. In section 5 we show that this semantics is basically equivalent to the semantics of the $\pi$-calculus.

We adhere to the following design decisions: during reduction, merging of nodes is performed only on the outer level. There is a clear seperation of levels (speaking in $\pi$-calculus terms: everything beyond a prefix or beyond a replication operator ! is on another level). As in $\pi$-calculus there is no reduction other than on the outermost level. Merging of channels which is done immediately in the $\pi$-calculus (by name substitution) is passed to other levels only when they emerge on the outer level during reduction.

**Definition 5 (Process Graphs)** *A* process graph *$P$ is inductively defined as follows: $P$ is a hypergraph where each edge $e$ is either labelled with $(k,n)Q$ where*

*Q is again a process graph,* $1 \leq n \leq ar(Q)$ *and* $1 \leq k \leq ar(e) - n$ *(e is a process waiting for a message with n ports arriving at its k-th node), with* $!Q$ *(e is a process which can replicate itself) or with the constant M (e is a message sent to its last node).*

*The structural congruence on process graphs is the smallest equivalence closed under isomorphism, graph construction and rule* (a) *below (replication uses the operator* $P \square Q = (P,\zeta) \otimes (Q,\zeta)$ *where* $\zeta : \mathbf{m} \twoheadrightarrow \mathbf{m}$ *and* $m = ar(Q) = ar(P)$*). The reduction relation (reception of a message and its nodes by a process) is generated by rule* (b) *and is closed under isomorphism and graph construction.*




# 5 Encoding the π-Calculus

In order to show how process graphs are related to the π-calculus we give an encoding, transforming process graphs into expressions in the π-calculus. We use the asynchronous polyadic π-calculus without choice which can be described by the following syntax (we assume that $N$ is a fixed set of names):

$$p := \mathbf{0} \mid (\nu c)p \mid \overline{c}\langle\tilde{a}\rangle \mid c(\tilde{x}).p \mid p_1|p_2 \mid !p \qquad c \in N, \tilde{a}, \tilde{x} \in N^*$$

The operational semantics of the π-calculus is defined as follows: structural congruence $\equiv_\pi$ is the smallest congruence closed under α-conversion and under the following rules:

$$p_1|p_2 \equiv_\pi p_2|p_1 \quad p_1|(p_2|p_3) \equiv_\pi (p_1|p_2)|p_3 \quad (\nu c)\mathbf{0} \equiv_\pi \mathbf{0} \quad !p \equiv_\pi !p|p$$
$$p|\mathbf{0} \equiv_\pi p \quad (\nu c)(\nu b)p \equiv_\pi (\nu b)(\nu c)p \quad ((\nu c)p_1)|p_2 \equiv_\pi (\nu c)(p_1|p_2),\ c \notin fn(p_2)$$

The reduction relation $\rightarrow_\pi$ satisfies $c(\tilde{x}).p \mid \overline{c}\langle\tilde{a}\rangle \rightarrow_\pi p\{\tilde{a}/\tilde{x}\}$ and is closed under parallel composition, hiding and structural congruence.

**Definition 6 (Encoding)** *Let P be a process graph and let* $\tilde{t} \in N^*$ *be a sequence without duplicates such that* $|\tilde{t}| = ar(P)$*. We define the encoding* $\Theta_{\tilde{t}}(P)$ *inductively as follows:*

**(Hypergraph)** $\Theta_{\tilde{t}}(\Delta_{\tilde{t}}(h)) = \Theta(h)$ **(Node/Empty Graph)** $\Theta(\langle a\rangle) = \Theta(0) = \mathbf{0}$

**(Replication)** $\Theta((!Q)[\tilde{t}]) = !\,[\Theta_{\tilde{u}}(Q)\{\tilde{t}/\tilde{u}\}]$ **(Message)** $\Theta((M)[\tilde{t}c]) = \overline{c}\langle\tilde{t}\rangle$

**(Process)** $\Theta(((k,n)Q)[\tilde{t}]) = \lfloor\tilde{t}\rfloor_k(\tilde{x}).[\Theta_{\tilde{u}}(Q)\{\tilde{t}\tilde{x}/\tilde{u}\}],\ \tilde{x} \in N^n$

**(Parallel Comp.)** $\Theta(h_1|h_2) = \Theta(h_1) \mid \Theta(h_2)$ **(Hiding)** $\Theta((\nu c)h) = (\nu c)\Theta(h)$

*where* $\tilde{u}, \tilde{x} \in N^*$ *are sequences of distinct fresh names.* $\Theta_{\tilde{t}}$ *is well-defined (up to isomorphism) because of proposition 1 and the definition of structural congruence.*

**Proposition 3** *Let $p$ be an arbitrary expression in the asynchronous polyadic $\pi$-calculus. Then there exists a process graph $P$ and a string $\tilde{t} \in N^*$ (without duplicates) such that $\Theta_{\tilde{t}}(P) \equiv_\pi p$. Furthermore for process graphs $P, P'$ and for every duplicate-free string $\tilde{t} \in N^*$ with $|\tilde{t}| = ar(P) = ar(P')$ it is true that:*

(a) *$P \equiv P'$ implies $\Theta_{\tilde{t}}(P) \equiv_\pi \Theta_{\tilde{t}}(P')$* (b) *$P \to P'$ implies $\Theta_{\tilde{t}}(P) \to_\pi \Theta_{\tilde{t}}(P)$*

(c) *$\Theta_{\tilde{t}}(P) \to_\pi p'$ implies the existence of $P'$ with $P \to P'$ and $\Theta_{\tilde{t}}(P') \equiv_\pi p'$*

The implication in (a) above does not hold for the opposite direction. This is due to several reasons: isolated nodes in the process graph which do not appear in its $\pi$-calculus counterpart, the order of nodes with respect to an edge which is not preserved by the translation, and delayed substitution. All this complicates the proof of the correctness of the encoding but gives an easier graph-based semantics.

By using a barbed congruence for process graphs (barbed congruence [8] is a bisimulation relation which can be defined more or less independently of specific process calculi) it is not hard to show that the encoding $\Theta_{\tilde{t}}$ is fully abstract with respect to barbed congruence (i.e. it preserves barbed congruence).

# 6 Fusions and Duplicates in the External Nodes

Up until now we demanded that the sequence of external nodes of a hypergraph may not contain any duplicates. The advantages of dropping this restriction are the following: if, in the right-hand side of a rewrite rule, a node appears twice in the sequence of external nodes, the rewriting step fuses these two nodes.

The question now is: are there any problems if we drop the restriction? The categorical construction operation can be defined as before, yet there are complications in the name-based notation. Consider the hypergraph $h = (A)[ab] \mid (B)[ab]$ and the rewrite rule $((A)[ab], (C)[aa])$. This rewrite rule should fuse the nodes of the edge labelled $B$, i.e. $h$ should reduce to $(C)[aa] \mid (B)[aa]$, which requires a non-local definition of a rewriting step.

The same problem arises when defining process calculi where processes are able to fuse two names on their own. The fusion calculus [11] solves the problem by having non-local reduction rules. One appropriate solution would be to give a name-based notation for hypergraphs with duplicates in their external nodes and to use the equivalence rules for the structural congruence of the corresponding process calculus.

# 7 Conclusion and Comparison to Related Work

We have presented a graph rewriting semantics for the $\pi$-calculus. While it has always been common knowledge that structural congruence corresponds to graph

isomorphism and it has been customary to draw processes as flow-graphs for visualization purposes, there are very few papers formalizing this connection, especially we do not know of any paper doing this by introducing a name-based notation for graphs. In [10] a term-oriented presentation of hypergraphs (similar to our name-based notation) is made, but no connection to process calculi is established.

[9] gives a concurrent (labelled transition) semantics based on graph rewriting for the $\pi$-calculus, where, in contrast to our semantics, nodes of the graph are labelled with channel names and the rewriting rules are rather complex. An encoding of the $\pi$-calculus into closed action calculi (which have a different categorical graph model than ours) is presented in [3], where the translation is less strict concerning the seperation of hierarchies. Graph notations for different systems of concurrent or interaction combinators (which are related to $\pi$-calculus in the same way combinator logic is related to the $\lambda$-calculus) are given in [12, 6]. Because of the simple nature of combinators, the resulting graphs are not hierarchical.

# References


[1] Gérard Berry and Gérard Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[2] H. Ehrig. Introduction to the algebraic theory of graphs. In *Proc. 1st International Workshop on Graph Grammars*, pages 1–69. Springer-Verlag, 1979. LNCS 73.

[3] Philippa Gardner. Closed action calculi. *Theoretical Computer Science (in association with the conference on Mathematical Foundations in Programming Semantics)*, 1998.

[4] Barbara König. Generating type systems for process graphs. In *Proc. of CONCUR '99*, pages 352–367. Springer-Verlag, 1999. LNCS 1664.

[5] Barbara König. Hypergraph construction and its application to the compositional modelling of concurrency. In *GRATRA '2000*, 2000.

[6] Yves Lafont. Interaction combinators. *Information and Computation*, 137(1):69–101, 1997.

[7] Robin Milner. The polyadic $\pi$-calculus: a tutorial. In *Logic and Algebra of Specification*. Springer-Verlag, 1993.

[8] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Proc. of ICALP '92*. Springer-Verlag, 1992. LNCS 623.

[9] Ugo Montanari and Marco Pistore. Concurrent semantics for the $\pi$-calculus. *Electronic Notes in Theoretical Computer Science*, 1, 1995.

[10] Ugo Montanari and Francesca Rossi. Graph rewriting and constraint solving for modelling distributed systems with synchronization. In *Coordination Languages and Models*, pages 12–27. Springer-Verlag, 1996. LNCS 1061.

[11] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proc. of LICS '98*, 1998.

[12] Nobuko Yoshida. Graph notation for concurrent combinators. In *Proc. of TPPP '94*. Springer-Verlag, 1994. LNCS 907.